Air Gap = Environnement froid (hors ligne)

Hot Node = nœud synchronisé (en ligne)

Cardano-cli:~$ Study sheets (Français)

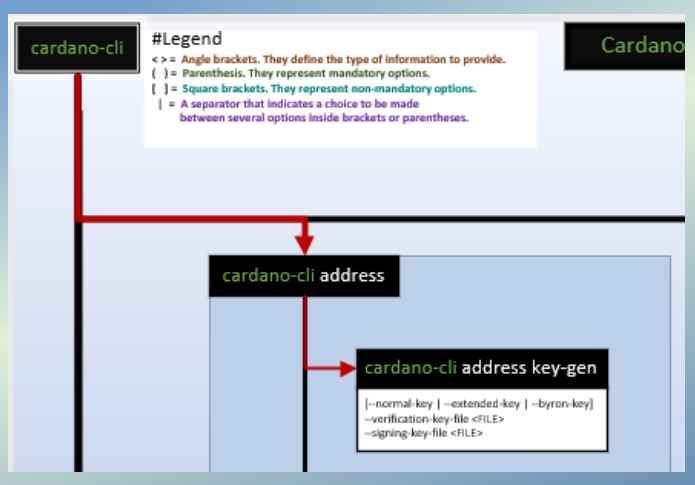
Partie 1 : Clés, adresse et délégation

Ce tutoriel est conçu pour être utilisé avec la version imprimable de la Cardano-cli cheat sheet V8.0.0

Ce document a pour but d'expliquer en détail comment interpréter les commandes cardano-cli et leurs options afin de pouvoir les assembler soi-même si nécessaire. Pour ce faire, vous devez disposer d'un ordinateur et y installer un nœud de la blockchain Cardano et l'interphase de ligne de commande Cardano (cardano-cli). Vous commencerez d'abord par des commandes simples et nous augmenterons en complexité au fur et à mesure que le tutoriel progresse.

Premier exercice : Création des clés de paiement et de mise — Air Gap

1 **Tout d'abord, localisez la branche que vous allez utiliser pour vos clés de paiement.**



2 **Vous avez 5 options au total.**

Les 3 premières options ont des crochets avec 2 séparateurs ce qui indique que le choix entre ces 3 options n'est pas obligatoire puisqu'une clé normale sera utilisée par défaut si rien n'est spécifié. Pour cet exemple, vous n'avez pas à les utiliser.

```
cardano-cli address key-gen
[--normal-key | --extended-key | --byron-key]
--verification-key-file <FILE>
--signing-key-file <FILE>
```

3 **Les 2 options suivantes doivent être utilisées.**

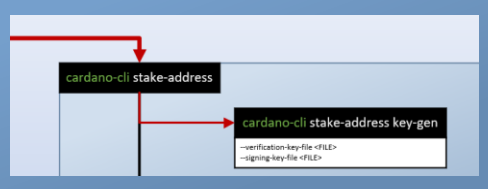
Les crochets angulaires indique le type d'information <FILE>. Vous devez fournir le nom que vous donnerez à vos 2 fichiers clés respectifs.

```
cardano-cli address key-gen
[--normal-key | --extended-key | --byron-key]
--verification-key-file payment.vkey
--signing-key-file payment.skey
```

4 **Ceci est le résultat final de cette commande simple sur votre terminal.**

```
user@computer:~$ cardano-cli address key-gen \
> --verification-key-file payment.vkey \
> --signing-key-file payment.skey
```

5 **Maintenant que vos clés de paiement sont prêtes, vous devez créer vos clés de mise. Cette commande est encore plus facile car il n'y a qu'un seul type de clé de mise.**




```
cardano-cli stake-address key-gen
--verification-key-file <FILE>
--signing-key-file <FILE>
```

6 **Tout comme 3 . Vous devez indiquer le type d'information. <File>**

```
cardano-cli stake-address key-gen
--verification-key-file stake.vkey
--signing-key-file stake.skey
```

7 **Et pour le résultat final sur le terminal...**

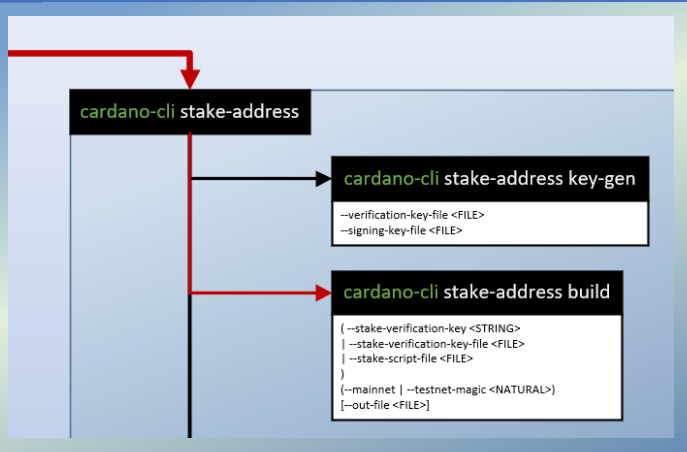
```
user@computer:~$ cardano-cli stake-address key-gen \
> --verification-key-file stake.vkey \
> --signing-key-file stake.skey
```

Avertissement. Il est recommandé de générer et d'utiliser vos clés de paiement et « stake key » pour signer des transactions dans un environnement "Air Gap" pour des raisons de sécurité. ***https://developers.cardano.org/docs/get-started/air-gap***

Maintenant que vos 2 paires de clés sont créées, vous allez pouvoir créer une adresse de mise qui vous permettra de vous renseigner sur le montant de vos récompenses et vous permettra de les retirer lors d'une transaction avec votre stake.skey.

Deuxième exercice : Création d'une adresse de mise — Air Gap

1 **Localisez la branche que vous allez utiliser pour créer votre adresse de mise.**



2 **Vous avez 6 options au total.**

Les 3 premières options sont jointes par des parenthèses et comportent 2 séparateurs qui indiquent un choix obligatoire à faire entre ces trois options.

```
cardano-cli stake-address build
( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
)
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

3 **Utilisez l'option stake-verification-key-file.**

Les crochets angulaires indique le type d'information <FILE>. Cette fois, vous devez fournir le chemin qui mène à votre stake.vkey

```
cardano-cli stake-address build
( --stake-verification-key <STRING>
| --stake-verification-key-file stake.vkey
| --stake-script-file <FILE>
)
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

4 **Vous avez maintenant 2 options entre parenthèses.**

Vous devez préciser le réseau utilisé et s'il s'agit du testnet, mentionner le numéro magique du réseau. Utilisons le réseau

```
cardano-cli stake-address build
--stake-verification-key-file stake.vkey
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

5 **Et pour la dernière option --out-file<FILE>**

Celui-ci n'est pas obligatoire car il est entre crochets. Mais si vous n'utilisez pas cette option, le « output » (adresse de mise) de la commande sera affichée dans votre terminal au lieu d'un fichier, et puisque vous aurez besoin d'utiliser cette adresse de mise plus tard, donnons-lui un nom.

```
cardano-cli stake-address build
--stake-verification-key-file stake.vkey
--mainnet
--out-file stake.addr
```

6 **Voici à quoi ressemblera cette commande sur votre terminal.**

```
user@computer:~$ cardano-cli stake-address build \
> --stake-verification-key-file stake.vkey \
> --mainnet \
> --out-file stake.addr
```

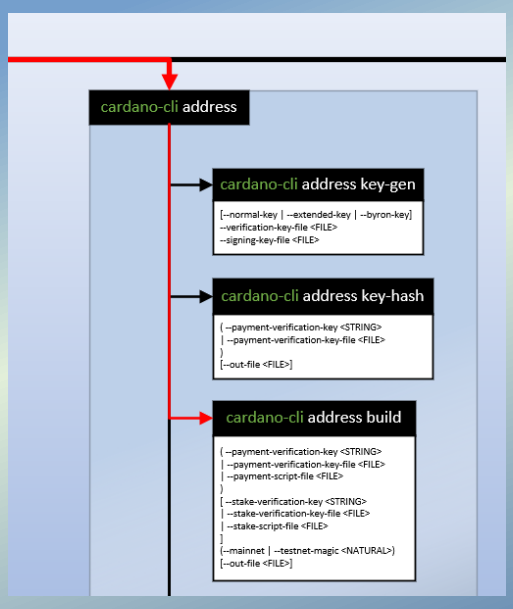
7 **Voici ce que vous devriez avoir jusqu'à présent.**

```
user@computer:~$ ls
payment.vkey    payment.skey    stake.vkey
stake.skey      stake.addr
```

Maintenant que vos 2 paires de clés et votre adresse de mise sont créées, vous allez pouvoir créer une adresse combinant votre clé de paiement avec la clé de mise afin que l'argent de l'adresse générée soit inclus dans le protocole de mise avec vos récompenses.

Troisième exercice : Création du fichier d'adresse « paymentwithstake.addr » — Air Gap

1 **Localisez la branche que vous allez utiliser pour votre « paymentwithstake.addr » .**



2 **Vous avez 9 options au total.**

Il vous est possible de créer une adresse de paiement en utilisant uniquement votre clé de paiement sans la lier à votre clé de mise, ce qui explique pourquoi le 1er groupe d'options est obligatoire et non le 2ème groupe.

cardano-cli address build

```
( --payment-verification-key <STRING>
| --payment-verification-key-file <FILE>
| --payment-script-file <FILE>
)
[ --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
]
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

3 **Utilisez à la fois vos clés de vérification de paiement et de mise**

Encore une fois, selon les crochets angulaires <FILE>, vous devez définir le chemin vers votre payment.vkey et votre stake.vkey.

cardano-cli address build

```
( --payment-verification-key <STRING>
| --payment-verification-key-file payment.vkey
| --payment-script-file <FILE>
)
[ --stake-verification-key <STRING>
| --stake-verification-key-file stake.vkey
| --stake-script-file <FILE>
]
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

4 **Vous allez maintenant mentionner le réseau à utiliser et le nom du fichier pour votre adresse.**

cardano-cli address build

```
--payment-verification-key-file payment.vkey
--stake-verification-key-file stake.vkey
(--mainnet | --testnet-magic <NATURAL>)
--out-file paymentwithstake.addr
```

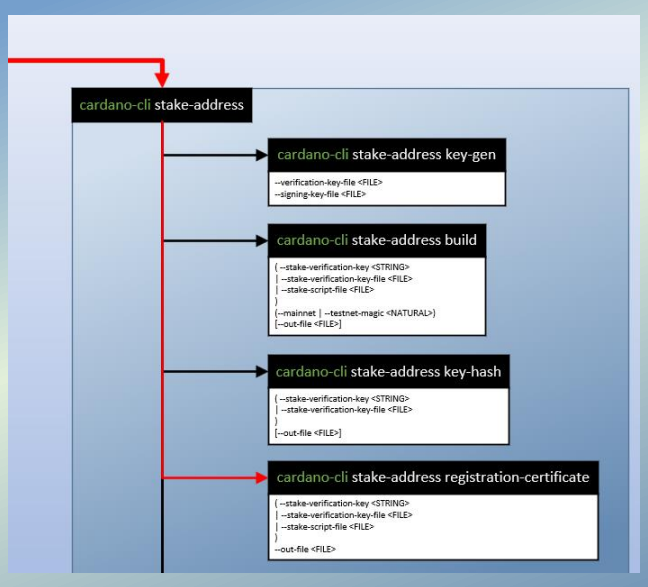
5 **C'est le résultat final.**

```
user@computer:~$ cardano-cli address build \
> --payment-verification-key-file payment.vkey \
> --stake-verification-key-file stake.vkey \
> --mainnet \
> --out-file paymentwithstake.addr
```

Vous pouvez copier le contenu de paymentwithstake.addr dans un éditeur de texte et ensuite l'utiliser comme adresse d'envoi dans le portefeuille Cardano que vous utilisez habituellement et lui envoyer de l'ada. (10 ada devraient suffire pour commencer)

Quatrième exercice : Création du certificat de mise — Air Gap

1 **Localisez la branche que vous allez utiliser pour votre certificat de mise.**



2 **Vous avez 4 options**

Pour pouvoir participer au protocole et jalonner votre ada, vous devez lier votre clé de vérification de mise à un certificat que vous soumettrez à la blockchain dans les prochains exercices. La commande pour créer votre certificat est assez simple. Il vous suffit de fournir l'une de ces 3 options obligatoires et de préciser le nom du fichier qui vous servira de certificat.

cardano-cli stake-address registration-certificate

```
( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
)
--out-file <FILE>
```

cardano-cli stake-address registration-certificate

```
( --stake-verification-key <STRING>
| --stake-verification-key-file stake.vrf
| --stake-script-file <FILE>
)
--out-file stake.cert
```

3 **Ceci est le résultat final de la façon dont cette commande devrait ressembler sur votre terminal.**

```
user@computer:~$ cardano-cli stake-address registration-certificate \
> --stake-verification-key-file stake.vkey \
> --out-file stake.cert
```

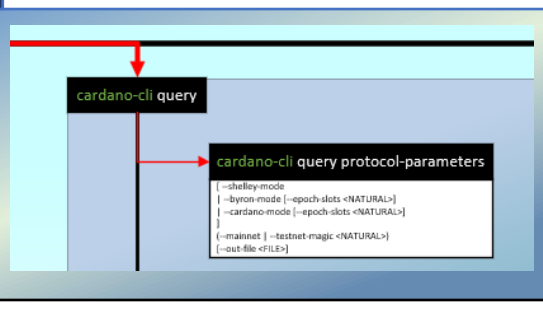
4 **Voici ce que vous devriez avoir jusqu'à présent**

```
user@computer:~$ ls
payment.vkey        payment.skey        stake.vkey        stake.skey
stake.addr          paymentwithstake.addr  stake.cert
```

Vous allez maintenant obtenir les paramètres du protocole et le numéro de slot actuel afin que vous puissiez commencer à construire votre toute première transaction.

Cinquième exercice : Obtention des paramètres du protocole — Hot Node

1 **Pour votre transaction, vous aurez besoin de paramètres de protocole pour le calcul des frais.**



2 **Vous avez 6 options et 2 sous-options au total.**

cardano-cli query protocol-parameters

```
[ --shelley-mode
| --byron-mode [--epoch-slots <NATURAL>]
| --cardano-mode [--epoch-slots <NATURAL>]
]
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

3 **Passer les options de mode. Mentionnez le réseau souhaité et le nom du fichier à créer.**

cardano-cli query protocol-parameters

```
[ --shelley-mode
| --byron-mode [--epoch-slots <NATURAL>]
| --cardano-mode [--epoch-slots <NATURAL>]
]
(--mainnet | --testnet-magic <NATURAL>)
[--out-file protocol.json
```

4 Ceci est le résultat final sur votre terminal.

```
user@computer:~$ cardano-cli query protocol-parameters \
> --mainnet \
> --out-file protocol.json
```

5 Maintenant, voyons le contenu de ce fichier :

```
user@computer:~$ cat protocol.json
```

Dans le fichier protocol.json vous allez chercher le dépôt à effectuer dans la blockchain pour enregistrer votre adresse de mise et participer au protocole de jalonnement. Ce dépôt peut être récupéré à tout moment si vous désenregistrez votre adresse.

6 Prenez note du montant du dépôt, car vous en aurez besoin plus tard. Le montant est en Lovelace. (1 ada = 1 000 000 Lovelace)

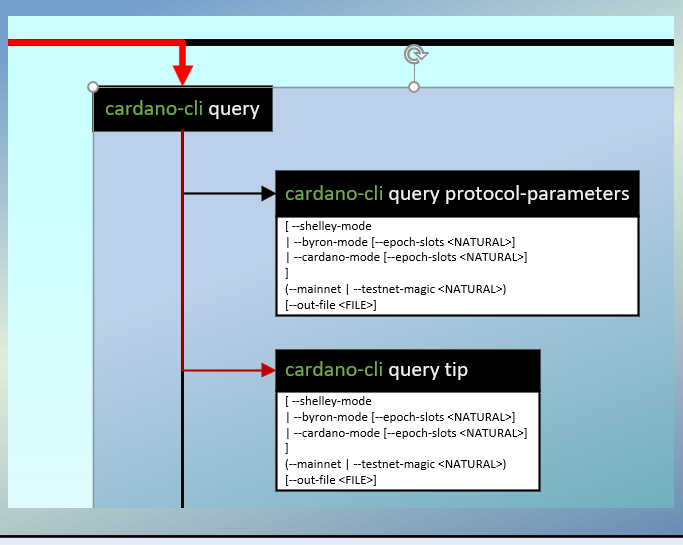
```
"poolRetireMaxEpoch": 18,
"protocolVersion": {
    "major": 8,
    "minor": 0
},
"stakeAddressDeposit": 2000000,
"stakePoolDeposit": 500000000,
"stakePoolTargetNum": 500,
"treasuryCut": 0.2,
"txFeeFixed": 155381,
"txFeePerByte": 44,
"utxoCostPerByte": 4310,
"utxoCostPerWord": null
```

7 Vous devez maintenant prendre votre fichier protocol.json et le transférer dans votre environnement "Air Gap" pour pouvoir calculer les frais.

⚠ **Avec le CIP-1694 et l'ère Voltaire qui est à nos portes, il sera possible pour les détenteurs d'ada de la communauté avec l'aide du comité constitutionnel et des Dreps de modifier les paramètres du protocole dans un système de vote bien élaboré. C'est pourquoi il est important de vous assurer d'avoir les dernières modifications de ce protocole dans votre environnement "Air Gap" car cela pourrait avoir un impact direct sur les différents paramètres entourant vos transactions.**

Sixième exercice : Obtention du numéro de slot — Hot Node

1 Vous devrez connaitre le numéro de slot actuel afin de calculer votre TTL. (description dans les exercices suivant)



2 Tout comme l'exercice précédent, 6 options 2 sous-options.

Maintenant que vous commencez à bien comprendre le principe, vous pouvez sauter quelques étapes. Pas besoin de créer un fichier, vous avez juste besoin du numéro de slot.

```
cardano-cli query tip
[ --shelley-mode
| --byron-mode [--epoch-slots <NATURAL>]
| --cardano-mode [--epoch-slots <NATURAL>]
]
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

3 Ceci est le résultat final sur votre terminal.

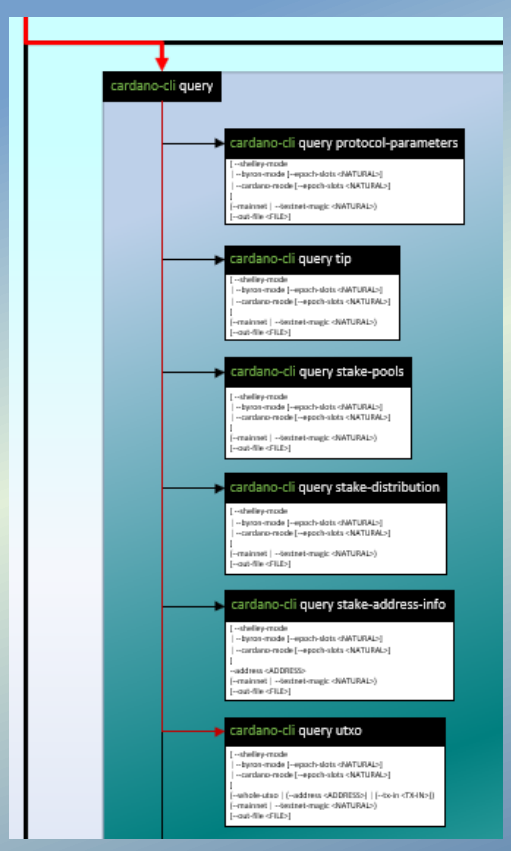
```
user@computer:~$ cardano-cli query tip \
> --mainnet
```

4 Notez le numéro de slot.

```
{
  "block": 8749125,
  "epoch": 410,
  "era": "Babbage",
  "hash": "503e4af96abc18e1d4b5de08e0d35cb508e364...",
  "slot": 92027764,
  "syncProgress": "100.00"
}
```

Seventh exercise: Query the UTXO — Hot Node

1 Vous allez maintenant réquisitionner les UTXO de votre paymentwithstake.addr.



2 Cette commande a 9 options 2 sous-options.

Vous devez utiliser au moins un UTXO comme « input » pour votre transaction. Une transaction peut contenir plusieurs « input » et plusieurs « output » mais dans ce cas, vous ne devez avoir qu'un seul UTXO associé à votre paymentwithstake.addr car vous n'avez effectué qu'un seul dépôt de 10 ada dans celui-ci. N'utilisons donc que ce qui est obligatoire. Bref, votre paymentwithstake.addr, le réseau à utiliser et le nom d'un fichier pour transporter cette liste d'UTXO vers votre environnement "air gap".

```
cardano-cli query utxo
[ --shelley-mode
| --byron-mode [--epoch-slots <NATURAL>]
| --cardano-mode [--epoch-slots <NATURAL>]
]
(--whole-utxo | (--address <ADDRESS>) | (--tx-in <TX-IN>))
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

3 Voici à quoi cela devrait ressembler sur votre terminal.

```
cardano-cli query utxo
[ --shelley-mode
| --byron-mode [--epoch-slots <NATURAL>]
| --cardano-mode [--epoch-slots <NATURAL>]
]
(--whole-utxo | (--address paymentwithstake.addr) | (--tx-in <TX-IN>))
(--mainnet | --testnet-magic <NATURAL>)
(--out-file UTXO.addrs)
```

```
user@computer:~$ cardano-cli query utxo \
> --address paymentwithstake.addr
> --mainnet
> --out-file utxo.addrs
```

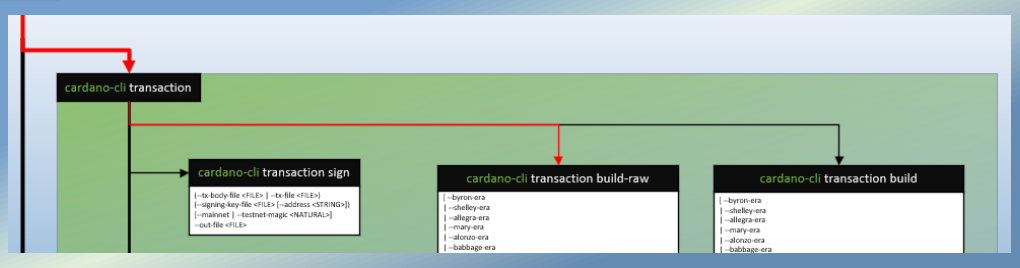
4 Le contenu du fichier utxo.addr devrait ressembler à ceci. L'UTXO de votre dépôt de 10 ada = 10 000 000 Lovelace. Vous pouvez maintenant prendre ce fichier et le mettre dans votre environnement "Air Gap". Vous en aurez besoin bientôt.

```
                           TxHash                                 TxIx        Amount
--------------------------------------------------------------------------------------
1234a4d18e9dkhb34234kjbvdec3ad81e299c1a523443453561e61ce9bf8608e8c802df3b7f8c     0        10000000 lovelace + TxOutDatumNone
```

Il est maintenant temps de construire votre première transaction qui servira à soumettre votre certificat de mise. Avant de commencer, ce qui vous attend pourrait sembler intimidant, mais au fur et à mesure que vous progresserez, vous devriez être en mesure de comprendre pourquoi et comment vous réduirez les prochaines options à 6 options au total pour votre processus de transaction. Pour des raisons de sécurité, dans ce tutoriel, vous utiliserez des méthodes impliquant la commande " cardano-cli transaction build-raw " au lieu de la commande " cardano-cli transaction build " car elle peut être construite dans un environnement hors ligne.

Eighth exercise: Creation of your first transaction draft — Air Gap

1 **Localisez la branche ''cardano-cli transaction build-raw''**



2 **Commençons progressivement du haut vers le bas.**

```
cardano-cli transaction build-raw
[ --byron-era
| --shelley-era
| --allegra-era
| --mary-era
| --alonzo-era
| --babbage-era
]
```

3 **Les 5 premières options sont facultatives [crochets] En ne mentionnant rien, ce sera l'ère Mary par défaut.**

```
cardano-cli transaction build-raw
[ --byron-era
| --shelley-era
| --allegra-era
| --mary-era
| --alonzo-era
| --babbage-era
]
[--script-valid | --script-invalid]
```

4 **Votre transaction n'implique pas de script, vous pouvez donc ignorer les 2 options suivantes.**

```
cardano-cli transaction build-raw
[ --byron-era
| --shelley-era
| --allegra-era
| --mary-era
| --alonzo-era
| --babbage-era
]
[--script-valid | --script-invalid]
```

5 **Ensuite pour les options suivantes et ses 20 sous-options, une explication s'impose.**

À l'intérieur d'une parenthèse d'option, il peut y avoir des sous-options définies par des colonnes. C'est pourquoi il existe une notion de priorité et un ordre particulier à respecter lors de la construction d'une transaction. Dans ce cas vous savez qu'il y a 3 colonnes distinctes définissant l'ordre dans lequel les options doivent être saisies si l'on veut que le corps de la transaction soit produit correctement.

```
1 [--script-valid | --script-invalid]
  (--tx-in <TX-IN>
2  [--spending-tx-in-reference <TX-IN>
    --spending-plutus-script-v2
3   ( --spending-reference-tx-in-datum-cbor-file
    | --spending-reference-tx-in-datum-file
```

6 **Prenez le temps de bien analyser l'ordre de priorité de l'option "tx-in" et de ses parenthèses.**

```
[--script-valid | --script-invalid]
(--tx-in <TX-IN>
 [ --spending-tx-in-reference <TX-IN>
  --spending-plutus-script-v2
  ( --spending-reference-tx-in-datum-cbor-file <CBOR FILE>
  | --spending-reference-tx-in-datum-file <JSON FILE>
  | --spending-reference-tx-in-datum-value <JSON VALUE>
  | --spending-reference-tx-in-inline-datum-present
  )
  ( --spending-reference-tx-in-redeemer-cbor-file <CBOR FILE>
  | --spending-reference-tx-in-redeemer-file <JSON FILE>
  | --spending-reference-tx-in-redeemer-value <JSON VALUE>
  )
  --spending-reference-tx-in-execution-units (<INT, INT>)
 | --simple-script-tx-in-reference <TX-IN>
 | --tx-in-script-file <FILE>
  [
   ( --tx-in-datum-cbor-file <CBOR FILE>
   | --tx-in-datum-file <JSON FILE>
   | --tx-in-datum-value <JSON VALUE>
   | --tx-in-inline-datum-present
   )
   ( --tx-in-redeemer-cbor-file <CBOR FILE>
   | --tx-in-redeemer-file <JSON FILE>
   | --tx-in-redeemer-value <JSON VALUE>
   )
   --tx-in-execution-units (<INT, INT>)]
 ])
[--read-only-tx-in-reference <TX-IN>]
[--tx-in-collateral <TX-IN>]
```

7 **''--tx-in'' est obligatoire mais pas ses sous-options.**

Les options suivantes ne sont pas obligatoires et sont utilisées pour les scripts plutus. Vous n'avez donc pas besoin de les utiliser pour notre transaction simple.

```
--tx-in <TX-IN>
[ --spending-tx-in-reference <TX-IN>
  --spending-plutus-script-v2
  ( --spending-reference-tx-in-datum-cbor-file <CBOR FILE>
  | --spending-reference-tx-in-datum-file <JSON FILE>
  | --spending-reference-tx-in-datum-value <JSON VALUE>
  | --spending-reference-tx-in-inline-datum-present
  )
  ( --spending-reference-tx-in-redeemer-cbor-file <CBOR FILE>
  | --spending-reference-tx-in-redeemer-file <JSON FILE>
  | --spending-reference-tx-in-redeemer-value <JSON VALUE>
  )
  --spending-reference-tx-in-execution-units (<INT, INT>)
 | --simple-script-tx-in-reference <TX-IN>
 | --tx-in-script-file <FILE>
  [
   ( --tx-in-datum-cbor-file <CBOR FILE>
   | --tx-in-datum-file <JSON FILE>
   | --tx-in-datum-value <JSON VALUE>
   | --tx-in-inline-datum-present
   )
   ( --tx-in-redeemer-cbor-file <CBOR FILE>
   | --tx-in-redeemer-file <JSON FILE>
   | --tx-in-redeemer-value <JSON VALUE>
   )
   --tx-in-execution-units (<INT, INT>)]
 ])
```

8 **Pour ce qui suit :**

Vous n'aurez pas non plus besoin de ''read only reference input'' ou quoi que ce soit concernant le collatéral puisqu'il s'agit d'une simple transaction qui n'inclura pas de script Plutus.

```
[--read-only-tx-in-reference <TX-IN>]
[--tx-in-collateral <TX-IN>]
[--tx-out-return-collateral <ADDRESS VALUE>]
[--tx-total-collateral <INTEGER>]
[--required-signer <FILE> | --required-signer-hash <HASH>]
[--tx-out <ADDRESS VALUE>
  [ --tx-out-datum-hash <HASH>
  | --tx-out-datum-hash-cbor-file <CBOR FILE>
```

9 **À propos de "--required-signer-hash <HASH>"**

Cette option ne sera pas utile à ce moment pour votre transaction qui servira à soumettre votre stake.cert. Mais il faut noter qu'elle vous sera très utile dans l'exercice à propos du vote de gouvernance.

```
[--read-only-tx-in-reference <TX-IN>]
[--tx-in-collateral <TX-IN>]
[--tx-out-return-collateral <ADDRESS VALUE>]
[--tx-total-collateral <INTEGER>]
[--required-signer <FILE> | --required-signer-hash <HASH>]
[--tx-out <ADDRESS VALUE>
```

10 **Enfin! Une option dont vous aurez besoin, "--tx-out"**

Vous aurez besoin de cette option pour spécifier l'adresse qui recevra le solde de votre UTXO consommé moins les frais. Alors copions cette option et ajoutons-la à votre brouillon de transaction.

```
[--read-only-tx-in-reference <TX-IN>]
[--tx-in-collateral <TX-IN>]
[--tx-out-return-collateral <ADDRESS VALUE>]
[--tx-total-collateral <INTEGER>]
[--required-signer <FILE> | --required-signer-hash <HASH>]
--tx-out <ADDRESS VALUE>
```

11 **Les sous-options de "tx-out" à propos du script plutus peuvent être ignorées.**

```
--tx-out <ADDRESS VALUE>
[ --tx-out-datum-hash <HASH>
| --tx-out-datum-hash-cbor-file <CBOR FILE>
| --tx-out-datum-hash-file <JSON FILE>
| --tx-out-datum-hash-value <JSON VALUE>
| --tx-out-datum-embed-cbor-file <CBOR FILE>
| --tx-out-datum-embed-file <JSON FILE>
| --tx-out-datum-embed-value <JSON VALUE>
| --tx-out-inline-datum-cbor-file <CBOR FILE>
| --tx-out-inline-datum-file <JSON FILE>
| --tx-out-inline-datum-value <JSON VALUE>
]
[--tx-out-reference-script-file <FILE>]]
[--mint <VALUE>
```

12 **Pas de multi-asset, pas de NFT, pas de script plutus pour vos transactions.**

```
[--mint <VALUE>
 ( --mint-script-file <FILE>
  [
   ( --mint-redeemer-cbor-file <CBOR FILE>
   | --mint-redeemer-file <JSON FILE>
   | --mint-redeemer-value <JSON VALUE>
   )
   --mint-execution-units (<INT, INT>)]
 | --simple-minting-script-tx-in-reference <TX-IN> --policy-id <HASH>
 | --mint-tx-in-reference <TX-IN>
  --mint-plutus-script-v2
  ( --mint-reference-tx-in-redeemer-cbor-file <CBOR FILE>
  | --mint-reference-tx-in-redeemer-file <JSON FILE>
  | --mint-reference-tx-in-redeemer-value <JSON VALUE>
  )
  --mint-reference-tx-in-execution-units (<INT, INT>)
  --policy-id <HASH>
 )]
```

13 **Vous utiliserez alors 3 des 4 options suivantes.**

- ''--invalid-before'' détermine à partir de quel Slot la transaction sera valide pour être traitée.
- tandis que ''--invalid-herafter'' détermine à partir de quel Slot la transaction deviendra invalide. (comme une date d'expiration)

```
[--invalid-before <SLOT>]
[--invalid-hereafter <SLOT>]
[--fee <LOVELACE>]
[--certificate-file <CERTIFICATEFILE>
```

14 **Prennons le TTL, les frais et le fichier de certificat.**

C'est pourquoi vous avez fait une ''cardano-cli query tip'' quelques exercices plus tôt. En connaissant le numéro de slot de votre nœud synchronisé, vous pouvez déterminer une ''durée de vie'' ou ''TTL'' pour votre transaction pendant qu'elle se trouve dans le pool de mémoire. Vous pouvez donc maintenant ajouter à votre brouillon ces trois options qui seront détaillées plus tard.

```
[--invalid-before <SLOT>]
[--invalid-hereafter <SLOT>]
[--fee <LOVELACE>]
[--certificate-file <CERTIFICATEFILE>
```

15 **Encore une fois, aucune option de certificat relative au script plutus ne sera utilisée.**

```
--certificate-file <CERTIFICATEFILE>
[ --certificate-script-file <FILE>
  [
   ( --certificate-redeemer-cbor-file <CBOR FILE>
   | --certificate-redeemer-file <JSON FILE>
   | --certificate-redeemer-value <JSON VALUE>
   )
   --certificate-execution-units (<INT, INT>)]
 | --certificate-tx-in-reference <TX-IN>
  --certificate-plutus-script-v2
  ( --certificate-reference-tx-in-redeemer-cbor-file <CBOR FILE>
  | --certificate-reference-tx-in-redeemer-file <JSON FILE>
  | --certificate-reference-tx-in-redeemer-value <JSON VALUE>
  )
  --certificate-reference-tx-in-execution-units (<INT, INT>)
 ]]
```

16 "--withdrawal" est un « input » qui vous permet de retirer les récompenses de votre stake.addr.

```
[--withdrawal <WITHDRAWAL>
 [ --withdrawal-script-file <FILE>
   [
    ( --withdrawal-redeemer-cbor-file <CBOR FILE>
    | --withdrawal-redeemer-file <JSON FILE>
    | --withdrawal-redeemer-value <JSON VALUE>
    )
    --withdrawal-execution-units (<INT, INT>)]
 | --withdrawal-tx-in-reference <TX-IN>
   --withdrawal-plutus-script-v2
   ( --withdrawal-reference-tx-in-redeemer-cbor-file <CBOR FILE>
   | --withdrawal-reference-tx-in-redeemer-file <JSON FILE>
   | --withdrawal-reference-tx-in-redeemer-value <JSON VALUE>
   )
   --withdrawal-reference-tx-in-execution-units (<INT, INT>)
 ]]
[--json-metadata-no-schema | --json-metadata-detailed-schema]
[--auxiliary-script-file <FILE>]
[--metadata-json-file <FILE> | --metadata-cbor-file <FILE>]
[--genesis <FILE> | --protocol-params-file <FILE>]
[--update-proposal-file <FILE>]
--out-file <FILE>
```

17 Vous pouvez l'ignorer ainsi que ceux liés au script plutus pour l'instant.

```
[--withdrawal <WITHDRAWAL>
 [ --withdrawal-script-file <FILE>
   [
    ( --withdrawal-redeemer-cbor-file <CBOR FILE>
    | --withdrawal-redeemer-file <JSON FILE>
    | --withdrawal-redeemer-value <JSON VALUE>
    )
    --withdrawal-execution-units (<INT, INT>)]
 | --withdrawal-tx-in-reference <TX-IN>
   --withdrawal-plutus-script-v2
   ( --withdrawal-reference-tx-in-redeemer-cbor-file <CBOR FILE>
   | --withdrawal-reference-tx-in-redeemer-file <JSON FILE>
   | --withdrawal-reference-tx-in-redeemer-value <JSON VALUE>
   )
   --withdrawal-reference-tx-in-execution-units (<INT, INT>)
 ]]
[--json-metadata-no-schema | --json-metadata-detailed-schema]
[--auxiliary-script-file <FILE>]
[--metadata-json-file <FILE> | --metadata-cbor-file <FILE>]
[--genesis <FILE> | --protocol-params-file <FILE>]
[--update-proposal-file <FILE>]
--out-file <FILE>
```

18 Plus que quelques options restantes.

Vous n'avez pas de métadonnées à soumettre pour le moment, ni de fichier de script auxiliaire, ni besoin de spécifier un fichier de paramètres **« genesis »** ou protocole. Et vous ne soumettez pas de proposition de mise à jour pour le fonds catalyst. Il ne vous reste plus qu'à nommer le fichier pour votre brouillon de transaction. (--out-fichier <FILE>)

```
[--json-metadata-no-schema | --json-metadata-detailed-schema]
[--auxiliary-script-file <FILE>]
[--metadata-json-file <FILE> | --metadata-cbor-file <FILE>]
[--genesis <FILE> | --protocol-params-file <FILE>]
[--update-proposal-file <FILE>]
--out-file <FILE>
```

19 En regroupant les options que vous avez copiées, vous obtiendrez quelque chose comme ceci :

cardano-cli transaction build-raw

```
--tx-in <TX-IN>
--tx-out <ADDRESS VALUE>
[--invalid-hereafter <SLOT>]
[--fee <LOVELACE>]
[--certificate-file <CERTIFICATEFILE>
--out-file <FILE>
```

20 Maintenant, pour terminer votre brouillon :

Vous ajoutez le UTXO (tx-in), l'adresse de change (tx-out) et le fichier de certificat. Pour l'instant, vous donnerez une valeur de 0 à tx-out, invalid-herafter et fee.

cardano-cli transaction build-raw

```
--tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0
--tx-out $(cat paymentwithstake.addr)+0
--invalid-hereafter 0
--fee 0
--certificate-file stake.cert
--out-file tx.raw
```

21 Voici à quoi cela ressemblera sur votre terminal :

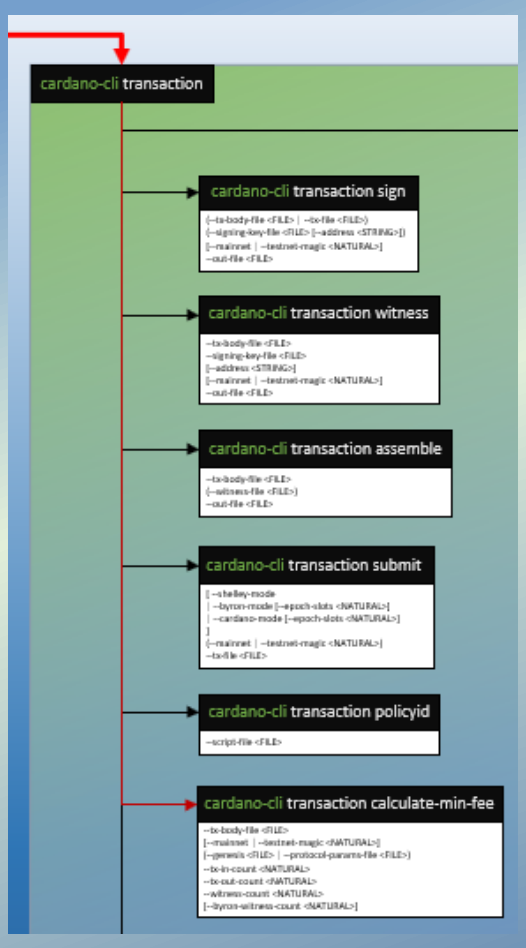
```
user@computer:~$ cardano-cli transaction build-raw \
> --tx-in 1234a4d18e9dkhb34234kjbvd81e299c#0 \
> --tx-out $(cat paymentwithstake.addr)+0 \
> --invalid-hereafter 0 \
> --fee 0 \
> --certificate-file stake.cert \
> --out-file tx.raw
```

Félicitations ! tu y es arrivé. Enregistrez la commande et les options de 21 dans un fichier d'éditeur de texte, vous en aurez besoin après le prochain exercice. Vous allez maintenant calculer les frais que vous coûtera votre transaction. Ensuite, vous pourez le soustraire du montant de votre UTXO (tx-in) et n'oubliez pas d'inclure le dépôt pour l'enregistrement de l'adresse de mise.

Neuvième exercice : Calcul des frais

Air Gap

1 Localisez la branche que vous allez utiliser pour le calcul de vos frais.



2 Vous avez 9 options au total.

Cette commande vous indiquera exactement le montant des frais que vous devrez débourser en fonction du nombre de tx-in, tx-out et du nombre de signatures requises.

cardano-cli transaction calculate-min-fee

```
--tx-body-file <FILE>
[--mainnet | --testnet-magic <NATURAL>]
(--genesis <FILE> | --protocol-params-file <FILE>)
--tx-in-count <NATURAL>
--tx-out-count <NATURAL>
--witness-count <NATURAL>
[--byron-witness-count <NATURAL>]
```

3 Seulement 3 de ces options ne seront pas utilisées.

- testnet-magic (évidemment, nous utilisons le mainnet pour ce tutoriel)
- genesis (nous utiliserons les paramètres de protocole)
- byron-witness-count (parce que vous n'utilisez pas de paires de clés byron)

cardano-cli transaction calculate-min-fee

```
--tx-body-file <FILE>
[--mainnet | --testnet-magic <NATURAL>]
(--genesis <FILE> | --protocol-params-file <FILE>)
--tx-in-count <NATURAL>
--tx-out-count <NATURAL>
--witness-count <NATURAL>
[--byron-witness-count <NATURAL>]
```

4 3 options à définir.

Précisons le nombre d'adresses de « input » et de « output » ainsi que le nombre de clés que vous utiliserez pour signer votre transaction.

cardano-cli transaction calculate-min-fee

```
--tx-body-file <FILE>
[--mainnet | --testnet-magic <NATURAL>]
(--genesis <FILE> | --protocol-params-file <FILE>)
--tx-in-count 1
--tx-out-count 1
--witness-count 2
[--byron-witness-count <NATURAL>]
```

5 Il vous suffit ensuite d'indiquer le PATH vers votre fichier protocol.json et votre brouillon de transaction tx.raw

cardano-cli transaction calculate-min-fee

```
--tx-body-file tx.raw
--mainnet
--protocol-params-file protocol.json
--tx-in-count 1
--tx-out-count 1
--witness-count 2
```

6 Voici le résultat dans votre terminal. (Le montant des frais ne sera pas toujours le même.)

```
user@computer:~$ cardano-cli transaction calculate-min-fee \
> --tx-body-file tx.raw \
> --mainnet \
> --protocol-params-file protocol.json \
> --tx-in-count 1 \
> --tx-out-count 1 \
> --witness-count 2
```

Si la commande fonctionne comme prévu, les frais apparaîtront en bas de celle-ci.

```
user@computer:~$ cardano-cli transaction calculate-min-fee \
> --tx-body-file tx.raw \
> --mainnet \
> --protocol-params-file protocol.json \
> --tx-in-count 1 \
> --tx-out-count 1 \
> --witness-count 2
178525 Lovelace
```

Pour le prochain exercice, vous devrez ouvrir le fichier de votre éditeur de texte que vous avez enregistré précédemment avec la commande "cardano-cli transaction build-raw" de l'exercice huit. Vous allez modifier son contenu pour construire votre transaction finale.

Dixième exercice : Construction de la transaction finale — Air Gap

1 — Ceci est votre brouillon de transaction de l'exercice huit

Vous le modifierez pour saisir le montant des frais (que vous connaissez) puis vous calculerez le montant de Lovelace à renvoyer à votre adresse.

cardano-cli transaction build-raw

```
--tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0
--tx-out $(cat paymentwithstake.addr)+0
--invalid-hereafter 0
--fee 178525
--certificate-file stake.cert
--out-file tx.raw
```

2 — En utilisant la commande "expr" vous pouvez effectuer votre calcul.

Montant du UTXO — Dépôt de l'adresse de mise — Frais

```
user@computer:~$ expr 10000000 - 178525 – 2000000
```

```
user@computer:~$ expr 10000000 - 178525 – 2000000
7821475
user@computer:~$
```

3 — Vous pouvez entrer le résultat dans votre transaction.

Notez qu'il ne doit y avoir aucun espace entre votre adresse, l'opérateur "+" et le montant en Lovelace. Sinon, il y aura une erreur lors de l'exécution de votre commande.

cardano-cli transaction build-raw

```
--tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0
--tx-out $(cat paymentwithstake.addr)+7821475
--invalid-hereafter 0
--fee 178525
--certificate-file stake.cert
--out-file tx.raw
```

4 — Maintenant, déterminons votre "TTL" (durée de vie)

Pour choisir à partir de quel Slot la transaction deviendra invalide, vous devez connaître le numéro de Slot dans lequel vous vous trouvez soit en répétant l'exercice #6 soit en vérifiant vos logs. Voici un exemple de ce que vous pourriez obtenir :

```
{
  "block": 8749178,
  "epoch": 410,
  "era": "Babbage",
  "hash": "367e4af96abc18e1d4b5de08af535cb508e691...",
  "slot": 92029934,
  "syncProgress": "100.00"
}
```

5 — Ajoutez-y quelques minutes. (1 slot = 1 seconde)

Pour vous permettre d'avoir le temps de signer votre transaction et de la soumettre sur votre "hot node", ajoutons 15 minutes à la valeur de l'option.
(92029934 + 900 = 92030834)

cardano-cli transaction build-raw

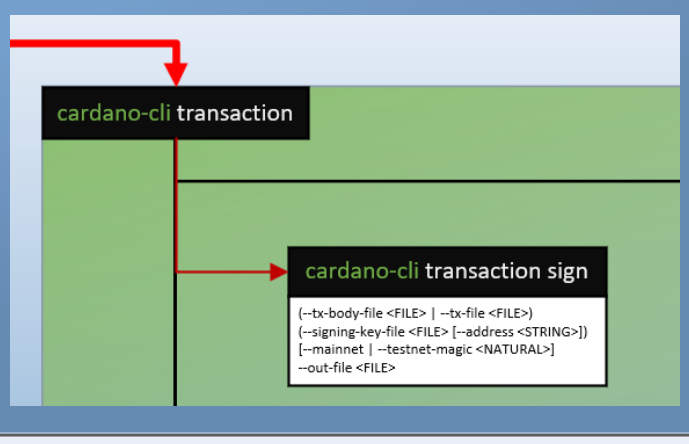
```
--tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0
--tx-out $(cat paymentwithstake.addr)+7821475
--invalid-hereafter 92030834
--fee 178525
--certificate-file stake.cert
--out-file tx.raw
```

6 — Voici le résultat dans votre terminal :

```
user@computer:~$ cardano-cli transaction build-raw \
> --tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0 \
> --tx-out $(cat paymentwithstake.addr)+7821475 \
> --invalid-hereafter 92030834 \
> --fee 178525 \
> --certificate-file stake.cert \
> --out-file tx.raw
```

Onzième exercice : Signature de votre transaction — Air Gap

1 — Vous êtes maintenant prêt à signer votre transaction avec vos 2 clés privées (payment.skey et stake.skey)



2 — Vous avez un total de 7 options.

Vous devrez mentionner votre "transaction body file", le chemin vers vos 2 clés privées, le réseau à utiliser et le nom du fichier que vous allez soumettre à la blockchain.

cardano-cli transaction sign

```
(--tx-body-file <FILE> | --tx-file <FILE>)
(--signing-key-file <FILE> [--address <STRING>])
[--mainnet | --testnet-magic <NATURAL>]
--out-file <FILE>
```

3 — Voici le résultat dans votre terminal :

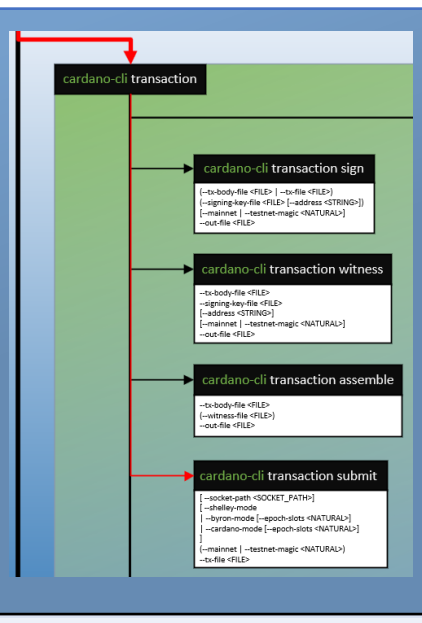
```
user@computer:~$ cardano-cli transaction sign\
> --tx-body-file tx.raw \
> --signing-key-file payment.skey \
> --signing-key-file stake.skey \
> --mainnet \
> --out-file tx.signed
```

⚠ **Notez que dans de nombreuses situations, certaines options peuvent être utilisées plusieurs fois.**

Vous pouvez maintenant transférer le fichier "tx.signed" vers votre "Hot Node" pour le soumettre à la blockchain mais assurez-vous d'abord que les permissions de celui-ci sont en "Read only".

Douzième exercice : Soumettre votre transaction — Hot Node

1 — Vous êtes maintenant prêt à soumettre votre transaction !



2 — Vous avez un total de 9 options.

L'option "--socket-path" n'est pas requise si le chemin vers votre fichier de socket est déjà dans votre environnement. Vous n'utiliserez que ce qui est nécessaire. Autrement dit, le réseau et le nom du fichier à soumettre.

cardano-cli transaction submit

```
[--socket-path <SOCKET_PATH>]
[--shelley-mode
| --byron-mode [--epoch-slots <NATURAL>]
| --cardano-mode [--epoch-slots <NATURAL>]
]
(--mainnet | --testnet-magic <NATURAL>)
--tx-file <FILE>
```

3 — Voici cette commande dans le terminal :

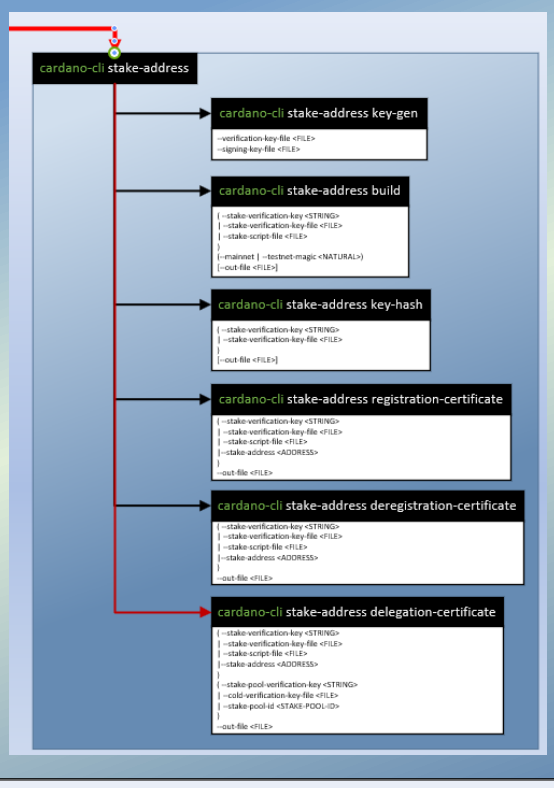
```
user@computer:~$ cardano-cli transaction submit \
> --mainnet \
> --tx-file tx.signed
```

```
user@computer:~$ cardano-cli transaction submit \
> --mainnet \
> --tx-file tx.signed
transaction successfully submitted
```

Félicitations, votre adresse de mise est maintenant enregistrée dans la blockchain. Vous pouvez désormais créer un certificat de délégation pour choisir un pool et participer au protocole "Proof of Stake" de Cardano. Cependant, avant de passer à l'autre exercice, assurez-vous de supprimer votre fichier tx.signed de votre "Hot Node". (Vous n'en aurez plus besoin)

Treizième exercice : Création d'un certificat de délégation — Air Gap

1 Tout d'abord, localisez la branche que vous allez utiliser pour votre certificat.



2 Vous avez 8 options au total.

Une fois soumis en chaîne, ce certificat sera utilisé pour indiquer avec quel pool vous jalonnerez votre ada. Il existe 2 groupes d'options obligatoires. Ces options seront choisies différemment selon vos besoins (ex : utiliser cold.vrf pour votre propre pool de mise) Dans cet exercice, nous supposons que vous souhaitez choisir entre plusieurs choix de pool.

```
cardano-cli stake-address delegation-certificate
( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
| --stake-address <ADDRESS>
)
( --stake-pool-verification-key <STRING>
| --cold-verification-key-file <FILE>
| --stake-pool-id <STAKE-POOL-ID>
)
--out-file <FILE>
```

3 Alors utilisons --stake-address et --stake-pool-id

- Mentionnez le chemin vers votre stake.addr.
- L'ID du pool de mise auquel vous souhaitez déléguer votre portefeuille. (peut être codé en Bech32 ou codé en Hex)
- Le nom de votre fichier de certificat.

```
cardano-cli stake-address delegation-certificate
( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
  --stake-address stake.addr

( --stake-pool-verification-key <STRING>
| --cold-verification-key-file <FILE>
  --stake-pool-id pool1mt8sdg37f2h3rypyuc77k7vxrjshtvjw04zdjlae9vdzyt9uu34
)
--out-file delegation.cert
```

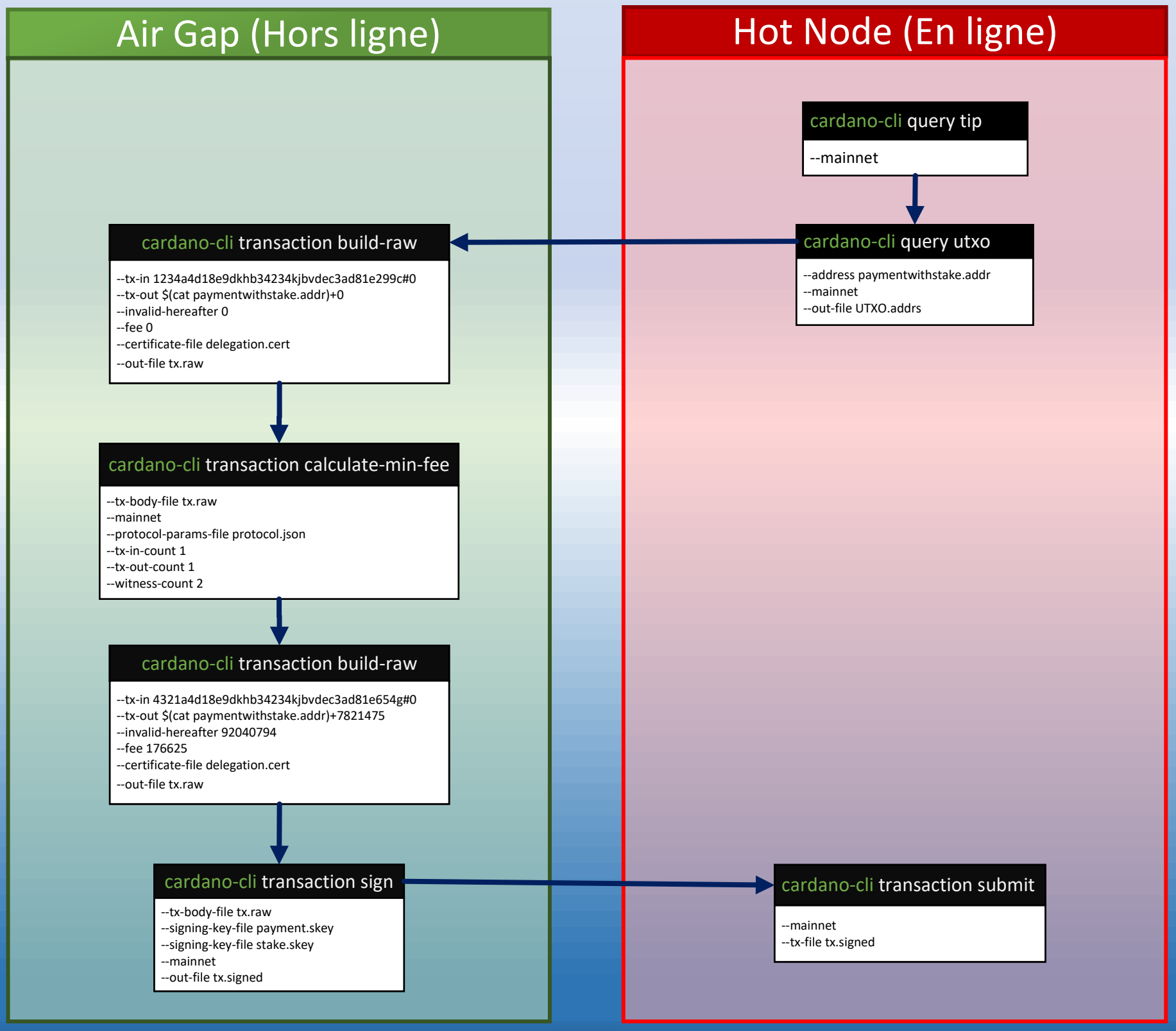
4 Voici le résultat dans votre terminal :

```
user@computer:~$ cardano-cli stake-address delegation-certificate \
> --stake-address stake.addr \
> --stake-pool-id pool1mt8sdg37f2h3rypyuc77k7vxrjshtvjw04zdjlae9vdzyt9uu34 \
> --out-file delegation.cert
```

⚠ Vous pouvez obtenir l'ID du pool de mise sur cexplorer.io ou si vous aimez ce document, faites-le nous savoir, nous pourrions ajouter les commandes : "query stake-pools", "query pool-state" et "query pool-distribution" au 2ème chapitre de ce tutoriel.

Vous pouvez maintenant répéter les exercices 6 à 12, en veillant à remplacer le stake.cert par le delegation.cert lorsque vous construisez votre transaction. Et n'oubliez pas que lors du calcul des frais, vous ne devez pas prendre en compte le dépôt d'adresse de mise. (ce qui a déjà été fait.)

Récapitulatif des opérations : Processus de soumission du certificat de délégation



Nous terminerons la partie 1 de ce tutoriel par une citation d'un collègue SPO que j'apprécie beaucoup :
« Nous devrions encourager les nouveaux SPOs, même s'ils sont moins qualifiés. Ils apprendront et Cardano décentralisera. »
--@StakeWithPride